Q56：合并区间

1. 合并区间时我们需要对左端点进行增序排序， 然后进行两两合并。
2. 直接使用快排，根据左端点排序。 然后使用第一个区间尝试和第二个区间合并。若第一区间和第二区间相离，那么第一区间就一定是孤立区间。
3. 直接加入结果集。 否则就合并。 合并结果成为新的第一区间。 再尝试合并第三。迭代进行。

Q57：插入区间： 一个无重叠的按左端点排序的区间列表，都是孤立区间。 现在插入一个给出的新区间，使得列表仍然无重叠。可以进行区间合并。

1. 按顺序比较，有交集就合并，无交集就加入ans,当出现第一个在新区间右侧的区间，且无交集时，把新区间的合并结果加入ans,后序都是无交集的，直接加入ans即可。

Q128：最长连续序列： 无序数组，找最长连续序列。[100,4,200,1,3,2] -> [1, 2, 3, 4] ans = 4; 时间要求o(n)

hard

1. 转换为区间问题

   ```
   我们使用leftmap(通过区间左边界，找右边界),rightmap(向反),记录每一个区间。
   假设现在有元素num ，那么我们把num当作左边界的元素，在leftmap中找左边界num+1，若能找到，则该区间可以扩张到num.
   然后把num当作右边界去扩张，如果num+1在leftmap，num-1也在rightmap，那么说名这两个区间可以合并。
   最大区间就是答案。
   ```

2. 将所有值放入 set中， 然后对set进行遍历， 对于set中每一个值，先去判断是否是一个完整区间的左端点，即set中不存在num-1这个数，那么num就是一个区间的起始，然后依次去判断是否存在num+i,i的大小就是区间长度。

Q435：无重叠区间： 删除最少的区间使得区间没有重合部分。

1. 肯定是先排序。 按照左端点排序。
2. 从右开始贪心，若两个区间重叠，则保留靠后的那个区间。
3. 为什么呢？ 贪心策略，靠后的区间 的左端点必然大，最不容易和前边的区间重叠。

Q452：用最少数量的箭引爆气球： 气球是就是一个映射在x轴的区间。从x轴垂直方向向上射箭。

0. 和 **Q435** 基本相同。

```
1．先按左端点排序。
2．从右开始。 last指向最后一个区间。然后从len-2开始遍历区间i，若i区间和last有重叠，那么i和last可以用一只箭。
3．若i-1仍然和last有重叠，那么 i-1,i,last可以用一只箭引爆。
4．直到i-k 和last无重叠。然后ans++,表示刚才几个区间用一个箭头，增加一只箭头
5．然后 last指向 i-k．重复刚才的过程。
```

Q632：最小区间，使得列表含有每个列表(非递减)至少其中一个数 也就是说，该最小区间必须满足，任意列表A，存在i 满足： L < A[i] < R。

0. 难点： 对于列表A[a1,a2,an]将A区间化：A[a1,an] 即使有区间 A包含最小区间，也无法说明 最小区间和A列表有交集，因为A列表是离散的，可能最小区间恰好只是覆盖了列表A没有元素的区域. 所以不能用最小右端点,和最大左端点作为最小区间.

1. 例子

```
1          5          20   21  ...
         4                 21  ...
            6              21  ...
              7            21  ...
                         20   21  ...
我们不能简单的认为最后一个列表的第一个元素就一定是最小区间的右端点．这个最小区间是[21,21]
1          5          20   21  ...
         4    7
            6              21  ...
              7            21  ...
                  19   20   21  ...
直观上,我们可能会选取,最小右端点7,和最大左端点19作为最小区间,但是是错误的.
```

2. 最小区间： 长度不同的区间，短的则是最小区间。 长度相同的区间，靠前的区间是最小区间。

3. 我们先选取一个可能的区间, 左端点为第一列表的第一元素, 右端点是最后一个列表的第一元素. 该区间可能不是最小的,还需要优化.但绝对和每个列表都有交集.

4. 然后将第一列表的左端点去除，由于列表左端点去除一个数，该列表左端点会变大。所以要重新按左端点排序。

5. 再取最末列表的左端点作为最小区间的右端点， 第一列表的左端点作为 最小区间的左端点。

6. 不断的迭代更新。

7. 由于迭代一次，就有一个列表减去一个元素。 如果一个列表减去一个元素为空了。 说明不能再迭代了。最小区间的左端点最大值找到了，再大，就与这个为空的列表没有交集了。

6 返回通过迭代更新的 最小区间。

7. 在该过程中，要不断的重新对 列表按照 左端点 排序， 所以 用 优先队列。

8. 当我们从队列中获取最小列表时，需要知道该 列表 是哪一个列表。该列表已经删除到第几个值了。所以设置 一个 Node

```
                Node{i(第几个列表),j(当前列表头是第几个元素),val(列表头元素的值)}
优先队列放Node
                大小根据 val值排序， 对Node的排序，其实就是对左端点的排序。
```

Q1024 : 视频拼接 : 视频片段 clips[i] 都用区间进行表示：开始于 clips[i][0] 并于 clips[i][1] 结束。我们甚至可以对这些片段

1. 自由地再剪辑，例如片段 [0, 7] 可以剪切成 [0, 1] + [1, 3] + [3, 7] 三部分.将剪辑后的内容拼接成覆盖整个运动过程的片段（[0, T]）。返回所需片段的最小数目.0 <= T <= 100 :这个暗示就很明显了。

2. 方法1： 记录 每个短视频 左端点相同的子区间 中最远的右端点，记为maxn[i]，即以i 开始的短视频中，其右端点 最远到 maxn[i]

```
       对于maxn[0]:代表以0开始的短视频中 最远可以播放到 maxn[0]. 我们记录这个值
pre = last. last = Math.max(last, maxn[i]);
       此时肯定这个段视频是必须要的。

               for (int[] clip : clips) {
                      if (clip[0] < T) {
                             maxn[clip[0]] = Math.max(maxn[clip[0]],
clip[1]);
                      }
               }
                for (int i = 0; i < T; i++) {
                      last = Math.max(last, maxn[i]);  // 每走到一个位置，就更
新最远能走到的位置。
                      if (i == last) {  如果i == last. last就是当前能走到的最远
的位置。   [[0,4],[5,8]] T=6   4和5是不连续的
                             return -1;
                      }
                       if (i == pre) { // 当一个pre到last，到pre时，就开启一个新
的短视频。
                             ret++;
                             pre = last;
                      }
               }
       注意：  [[0,4],[5,8]] T=6   4和5是不连续的。中间少了一秒，所以无法播放到
T=6.
```

3. 方法3：动态规划： dp[i]:播到时间i需要的视频片段数量。 那么dp[i]可能由dp[j] j+time >= i j < i 的时刻转换而来。

```
                for (int i = 1; i <= T; i++) {
                      for (int[] clip : clips) {
                             //  该区间可以覆盖 到 i ，或者覆盖范围超出 i 则可得
到 dp[i]
                             // dp[i] 是从 dp[clip[0]] 转移来的。 因为可能有很多
dp[clip[0]]都可以转移到 dp[i]
                             // 所以 这里 另 dp[i] 保持最小值。
                             // 可以播放到达 i 的短视频，都可以来更新他。
                             //该短视频的开始时间必然小于 i，终止时间 >= i即可。
                             if (clip[0] < i && i <= clip[1]) {
                                    dp[i] = Math.min(dp[i], dp[clip[0]] + 1);
                             }
                      }
                 }
```